Appendix A--A Step in the Dual Newton Method
Aleksandar Donev, 12/17/00

This Maple worksheet illustrates the basic step in the Dual Newton Method for convex network optimization when the arcs have a superconductor-like cost function.

> restart
> with(linalg)
> with(plots)

Part I: Making the network node-arc incidence matrix

The actual node-arc incidence matrix of this graph is composed of two parts:

- The basis part corresponding to a spanning tree:

\[
A_B := \begin{bmatrix}
1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & -1 & -1 \\
0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
-1 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

- A non-basis part corresponding to the non-tree arcs:

\[
A_N := \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & -1 & 0 \\
1 & 0 & -1 \\
0 & 0 & 1 \\
-1 & 0 & 0
\end{bmatrix}
\]

And the full matrix is the concatenation of the two:

\[
A_G := \text{concat}(A_B, A_N)
\]

This matrix has rank 6 even though it has 7 rows. So we should throw away one row corresponding to the root node:

> rank(A_G)

6

Take Root to be 1:

> Root := 1

So the actual node-arc incidence matrix we use is:

\[
A := \text{delrows}(A_G, \text{Root} .. \text{Root})
\]

\[
A := \begin{bmatrix}
0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & -1 \\
-1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0
\end{bmatrix}
\]
Part II) Cost and conjugate functions and derivatives:

In the case of a superconductor, the cost function is:

\[ f := (x, k) \rightarrow \frac{1}{2} R_k \xi_k^2 \left( \text{dilog(e}^{\frac{2|t-j_k|}{\xi_k}} + 1 \right) - \text{dilog(e}^{\frac{-2j_k}{\xi_k}} + 1) \right) + R_k \xi_k \ln(e^{\frac{2|t-j_k|}{\xi_k}} + 1) \mid x \]

It's derivative \( \frac{\partial}{\partial x} f(x) \) (the voltage \( V(I) \)) is:

\[ Df := (x, k) \rightarrow \text{evalf(signum(x) } R_k \mid x \left( 1 + \tanh \left( \frac{|x-j_k|}{\xi_k} \right) \right) \) \]

The second derivative \( \frac{\partial^2}{\partial x^2} f(x) \) is:

\[ D2f := (x, k) \rightarrow R_k \left( 1 + \tanh \left( \frac{|x-j_k|}{\xi_k} \right) \right) + \frac{R_k \mid x \left( 1 - \tanh \left( \frac{|x-j_k|}{\xi_k} \right) \right)^2}{\xi_k} \]

And the most important one is the inverse of the voltage-current function, \( (f'^{-1})(t) \), which involves the LambertW function:

\[ \text{invDf} := (t, k) \rightarrow \text{evalf} \left( \text{signum(t) } \frac{1}{2 R_k} \ln \left( \frac{2j_k - |t| R_k}{\xi_k} \right) \right) \]

The network cost function is the sum of the costs over all arcs:

\[ F := x \rightarrow \text{VecSum(Invoke(x,f))} \]

Part III) Random values for parameters:

The source vector:

\[ ST := [ \text{seq([.01 randvector(9)], } i = 1 \ldots n) ] \]

\[ ST := [.85, .57, .54, .01, .94, .49] \]

Resistances, critical currents and transition widths:

\[ R := [ \text{seq([.01 randvector(9)], } i = 1 \ldots m) ] \]

\[ R := [.99, .19, .23, .61, .17, .04, .62, .10, .14] \]
Part IV) Vectors and matrices used in the optimization

- The supply-demand vector:
  ```
  > b := evalm(ST)
  ```

- For now, we do not assign a specific value to the vector of Lagrange multipliers (voltages):
  ```
  > λ := vector(n)
  ```

- The tension vector (potential drops across arcs):
  ```
  > t := multiply(transpose(A), λ)
  ```

- The flow vector (currents) for the given potentials can now be found as $x_* = (f^{(-1)})(t)$:
  ```
  > x_* := Invoke(t, invDf)
  ```

- And the gradient is:
  ```
  > G := evalm(multiply(A, x_*) - b)
  ```

- The diagonal of the conjugate Hessian is the inverse of the diagonal of the cost-function Hessian:
  ```
  > DH_* := Diagonal(Invoke(x_*, (x, i) → 1/D2f(x, i)))
  ```

- And the full Hessian is:
  ```
  > H := multiply(A, DH_, transpose(A))
  ```

Part IV) Taking a step toward the minimum

This piece of code would be repeated until convergence to the global minimum.

We start by assigning a random vector of voltages to the vector of Lagrange multipliers:

```
> λ := evalm(V)
```

The direction vector is the solution to the Newton system $H d = -G$:

```
> d := linsolve(map(eval, H), evalm(-G))
```

Now we do a line search along this direction $λ = λ + α d$:

```
> λ := evalm(V + α d)
```

The objective function is the Lagrangian $h(α) = Lα + α.d$, where $Lα = x_*^T t - F(x_*) - λ^T d$:

```
> Lα := unapply(dotprod(map(eval, x_*), map(eval, t), 'orthogonal') - F(map(eval, x_*)))
```

This function should be concave and continuous:

```
> plot(Lα(α), α = 0 .. 1, numpoints = 10, adaptive = false, labels = ["alpha", "L[lambda](alpha)"])
```

To find the minimum (i.e. perform the line search), we find the derivative $h(α) = \frac{∂}{∂α} Lα(α)$, where
\[ h(\alpha) = d^T G, \] and set it to zero:

\[ h := \text{unapply}\left( \text{dotprod}(d, \text{map}(\text{eval}, G, \text{'orthogonal'}), \alpha) \right) \]

The plot below shows that \( h(\alpha) \) is not continuous. But this is an optical illusion, it comes from the fact that there is a very sharp non-linear transition in the \( V(I) \) curve. The derivative of this function

\[ \frac{\partial}{\partial \alpha} h(\alpha) = d^T H d \]

is this practically undefined (infinite) at certain points. The theory guarantees that \( h(\alpha) \) is piecewise convex, continuous, and non-decreasing. So Newton's method for the line search minimization, if used, must be used with great care and in combination with secant-like or bisection methods!

\[ dh := \text{unapply}\left( \text{dotprod}(d, \text{multiply}(\text{map}(\text{eval}, H), d, \text{'orthogonal'}), \alpha) \right) \]

\[ \text{plot}(h(\alpha), \alpha = 0 .. 1.5, \text{labels} = [ \text{"alpha"}, \text{"h(alpha)"} ]) \]

Now we find the zero of \( h(\alpha) \)--Beware, fsolve often fails here:

\[ \alpha := \text{fsolve}(h(\alpha) = 0, \alpha = 0 .. 2) \]
\[
\alpha := .3912935032
\]
And the new estimate for the potentials is:

\[
V := \text{map}(\text{eval}, \lambda)
\]
\[
[.6929396034, 1.006413019, .6061870287, .001976859083, .6572920878, .2618174621]
\]
This of course has not converged yet.